AMENDMENTS TO THE SPECIFICATION

Please amend the paragraph beginning at page 1, line 13, to read as follows:

There exists a general need in the development of software/systems to ensure that the finished product is sufficiently tested prior to its release to the public. Such testing is performed to detect programming errors/flaws. Corrections are formulated and then incorporated into subsequent versions of the software. There are various levels of testing thoroughness. The more ~~thorough~~ thoroughly the software is tested prior to release of the software to users, the less likely bugs will be exposed in the subsequent use of the released software.

Please amend the paragraph beginning at page 3, line 9, to read as follows:

In accordance with the present invention, ~~an API call replay tool~~ a tool for replaying an API call (hereinafter "API replay tool") facilitates creating and submitting API calls based upon input API call records. The API [[call]] replay tool ensures that the API calls, executed from potentially a sequence of logged API calls, replay in a meaningful manner. This is generally accomplished by translating addresses (memory references) specified by recorded API calls into addresses within the API [[call]] replay tool's memory space. More particularly, the API [[call]] replay tool includes a symbol table for mapping memory references within an input API call record into a memory space allocated to the API [[call]] replay tool. In a particular embodiment, such mapping occurs from a recorded address to a ~~replay~~ address space allocated to a thread within the API replay tool with which the API call is associated.

Please amend the paragraph beginning at page 3, line 18, to read as follows:

After mapping the ~~addresses~~ addresses/memory references into the ~~replay space~~ memory space of the API tool, an API call builder utilizes the address mapping relations stored within the symbol table to create a call code sequence for invoking the API ~~call in~~ to the ~~replay~~ environment of the API replay tool. The memory references within the [[API]] call code sequence are specified according to a set of mapping entries within the symbol table.

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

Please amend the paragraph beginning at page 5, line 11, to read as follows:

In summary of the system disclosed herein below, the API replay tool is a core component within a test system that takes as input a set of [[input]] API call records (described with reference to FIG. 3). The [[input]] inputted API call records include both the API call as well as information sufficient to enable re-creation of the environment within which the API is to be executed when the API call is replayed. While the API call records can be logged by an API call capture mechanism during execution of an application program, the source of the API call records is not material to the API replay tool.

Please amend the paragraph beginning at page 6, line 4, to read as follows:

Thus, as can be seen from the above summary, in an embodiment of the present invention, the API replay tool receives as input a sequence of API call records. The API replay tool creates a context for a thread within which the API call will be executed. ~~References~~ Memory references within the original API call are mapped to the memory space of the API replay tool. An API call is created and executed according to the context and mapping created by the API replay tool. Thus, there is no need for a tester to create an executable for submitting a sequence of API calls to perform a test of an API interface. Instead, the API replay tool itself is the executable and the API call records comprise data that drives the ~~executable~~ execution of the API replay tool.

Please amend the paragraph beginning at page 6, line 14, to read as follows:

Turning to the drawings, **FIG. 1** illustratively depicts an example of a suitable operating environment 100 for carrying out portions of the ~~application program interface~~ API replay tool and method embodying the present invention. The operating environment 100 is only one example of a suitable operating environment and is not intended to suggest any limitation as to the scope of use or functionality of the invention. Other well known computing systems, environments, and/or configurations that may be suitable for use with the invention include, but

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

are not limited to, personal computers, server computers, laptop/portable computing devices, hand-held computing devices, multiprocessor systems, microprocessor-based systems, network PCs, minicomputers, mainframe computers, distributed computing environments that include any of the above systems or devices, and the like.

Please amend the paragraph beginning at page 8, line 15, to read as follows:

The computer 110 may also include other removable/non-removable, volatile/nonvolatile computer storage media. By way of example only, **FIG. 1** illustrates a hard disk drive 140 that reads from or writes to non-removable, nonvolatile magnetic media, a magnetic disk drive 151 that reads from or writes to a removable, nonvolatile magnetic disk 152, and an optical disk drive 155 that reads from or writes to a removable, nonvolatile optical disk 156 such as a CD ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media that can be used in the exemplary operating environment include, but are not limited to, magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The hard disk drive 141 is typically connected to the system bus 121 through [[an]] a non-removable memory interface such as interface 140, and magnetic disk drive 151 and optical disk drive 155 are typically connected to the system bus 121 by a removable memory interface, such as interface 150.

Please amend the paragraph beginning at page 8, line 28, to read as follows:

The drives and their associated computer storage media discussed above and illustrated in **FIG. 1**, provide storage of computer readable instructions, data structures, program modules and other data for the computer 110. In **FIG. 1**, for example, hard disk drive 141 is illustrated as storing operating system 144, application programs 145, other program modules 146, and program data 147. Note that these components can either be the same as or different from operating system 134, application programs 135, other program modules 136, and program data 137. Operating system 144, application programs 145, other program modules 146, and program data 147 are given different numbers here to illustrate that, at a minimum, they are different copies. A user may enter commands and information into the computer 20 through input devices such as a keyboard 162 and pointing device 161, commonly referred to as a mouse,

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206 682 8100

trackball or touch pad. Other input devices (not shown) may include a microphone, joystick, game pad, satellite dish, scanner, or the like. These and other input devices are often connected to the processing unit 120 through a user input interface 160 that is coupled to the system bus, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). A monitor 191 or other type of display device may also be connected to the system bus 121 via an interface, such as a video interface 190. In addition to the monitor 191, computers may also include other peripheral output devices such as speakers 197 and printer 196, which may be connected through [[a]] an output peripheral interface 190.

Please amend the paragraph beginning at page 13, line 21, to read as follows:

The API replay tool 200 is intended to support an extensible set of special handlers that are selectively/conditionally invoked to ~~handler~~ handle particular API call scenarios. One such special case involves callback functions. Some API methods/functions support callback functionality. When such APIs are called from an application, the application passes a callback address for a function within the application's code space. When a particular condition/event is encountered, the operating system calls the function within the application's code space. Replaying such APIs requires establishing and passing an address for the callback function in the API call in the context of the replay tool 200, and then executing the callback function when it is invoked by the return call from the operating system. In an embodiment of the invention, the callback handler 220 processes [[-]] API calls that include the callback functionality. In such instances, a callback address is established for the callback function. The callback handler 220 receives the callback call when the operating system invokes the callback address. Thereafter, the callback handler 220 replays the set of APIs associated with the callback function through calls to the API call builder 222. The set of API calls associated with the callback function are obtained by/for the callback handler 220 from the set of API calls provided by the input file 206. Those calls are then sequentially submitted by the callback handler 220 to the API call builder 222 to facilitate their construction and execution within the context of the API replay tool 200. The callback ~~handle~~ handler 220 includes callback tracing mechanisms for handling callback functions invoked within callback functions. In each instance, the callback function

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206 682 8100

assigns a depth to the callback API call sequence and marks the return point in the previous level that invoked the callback function.

Please amend the paragraph beginning at page 14, line 28, to read as follows:

As mentioned previously above, in an embodiment of the invention, the API call builder 222 populates, and thereafter accesses the mapping information contained in, the symbol tables 214 that map memory references from their original logged addresses to their replay addresses. Thus, for each memory reference identified in an input API call record, the API call builder 222 creates a corresponding (mapped) location within the process space of the API replay tool 200 maintained by the memory manager 218. In an embodiment of the invention, the symbol tables 214 comprise a set of objects that maintain data structures facilitating mapping memory/names from the application 202 environment (i.e., before "replay") to the API replay tool 200 environment (i.e., at/after "replay"). Take for example the following example involving a handle. When CreateFile and ReadFile APIs are recorded, CreateFile returns a handle that ReadFile uses as an input parameter. For example:

0x12345678 = CreateFile(_,_,_,.....)

ReadFile(0x12345678,_,_,.....)

Please amend the paragraph beginning at page 16, line 5, to read as follows:

During the course of building an API call, the API call builder 222 obtains context information from the memory manager 218 for the particular thread with which the call is associated and extracts the relevant information for the API call (e.g., parameter values). The memory references (e.g., pointers) are the raw references provided in the original API call retrieved from the input file 206. Therefore, during the course of building an API call, the API call builder 222 accesses the symbol tables 214 to obtain the current memory locations of the referenced data. During the translation process that renders a sequence of assembly code instructions corresponding to an executable API call, the form of the assembly code generated by the API call builder 222 is based upon the system that will execute it (e.g., 32-bit system). The API call builder 222 stores the resulting sequence of assembly code instructions within a code segment of a memory block. The memory block within which the assembly code instructions are stored corresponds to a thread identified in the original API call record.

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206 682 8100

Please amend the paragraph beginning at page 17, line 1, to read as follows:

It is noted that 0x34343434 has replaced the string pointer value 0x12121212. This has occurred due to the translation, by the API call builder [[22]] 222, of the original address specified by the input API call record into the context of the API replay tool 200. The API call builder 222 recognized the parameter value 0x12121212 as being a pointer in the context of "recording" the API call. The value referenced by the string pointer value 0x12121212, "Hello," was also recorded by the API call logger 208. When the input file 206 was processed by the API replay tool, the "Hello" string was read into a dedicated location (x34343434) in the replay context (and more particularly the data segment), and the old and new pointer values were stored in the symbol tables 214. When the API call builder 222 creates assembly code for replaying the "foo" API call it accesses the symbol tables 214 and substitutes the reference value 0x12121212 with the replay context-based pointer value x34343434. Thus, when the assembly code sequence is executed, the pointer will properly reference the "Hello" string in the replay context rather than the original "recording" context.

Please amend the paragraph beginning at page 17, line 15, to read as follows:

After translating the API call record into an API call in the form of assembly code and data (in the replay context), the API call builder 222 passes the address of the assembly code, stored in the code segment of a particular thread's memory block, to an API call executer 224 to initiate execution of the API call by an operating system component (or some other component that executes the call such as an emulator). The API call executer 224 thereafter executes the passed assembly code instruction sequence and in doing so submits the API call built by the API call builder [[22]] 222 the operating system (or simulator/emulator thereof) component, such as DLL1 that supports execution of the called method or function of the operating system.

Please amend the paragraph beginning at page 17, line 24, to read as follows:

In an embodiment of the invention, the execution stage is carried out by the API call executer 224 that executes within a re-created execution environment (e.g., within the presence of a set of resources) of a particular binary file supplied by the Template DLL 226. The API call executer 224 receives a reference to the API call assembly code segment created by the API call

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

builder 222, executes the call embodied in the assembly code and associated data contained in an associated thread memory block, and then returns control back to the replay engine 212 for execution of a next API call record (or set of API call records).

Please amend the paragraph beginning at page 18, line 1, to read as follows:

By way of example, in an embodiment of the invention, the Template DLL 226 is a template from which multiple copies are made to simulate execution of a set of running binaries through injection of resources supplied by those binaries. Take for example, an application like "solitaire.exe" that has a cards.dll, and both solitaire.exe and cards.dll each has its own resources. Furthermore, a log file has the following sequence of records:

(1) Solitaire.exe Foo1(a,b)

(2) Solitaire.exe Foo2(c,d)

(3) Cards.dll Foo3(e,f)

(4) Cards.dll Foo4(g,h)

If log record (1) is going to be replayed, then a copy of the base Template DLL (e.g., Template DLL 226) is instantiated and named Solitaire.exe, and Foo1(a,b) is executed in the API call executer (e.g., API call executer 224) for Solitaire.exe. The record (2) is executed in the API executer 224 for Solitaire.exe as well. However, when record (3), which references ~~cards.dll~~ Cards.dll is executed, another copy of the Template DLL is created (not shown in FIG. 2) and named ~~cards.dll~~ Cards.dll. Thereafter, record (3) for an API Foo3(e,f) is called from ~~cards.dll~~ Cards.dll. Thus, in an embodiment of the invention, each Template DLL instance represents a distinct binary environment (e.g., resources) for executing API calls via the particular binary.

Please amend the paragraph beginning at page 18, line 27, to read as follows:

Turning to **FIG. 3** a set of fields associated with an API call are depicted. As mentioned previously herein above, in an embodiment of the invention the API replay tool 200 submits a sequence of API calls previously trapped during execution of an application. In such case a set of values are recorded that represent the state of the system at the time each API call was

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

executed to facilitate, to the extent possible, re-creating the system state when the API call is replayed through the API replay tool 200.

Please amend the paragraph beginning at page 19, line 1, to read as follows:

In an embodiment of the invention, the [[API]] log reader 210 supports API call records specifying replay/context information including: API call 300, API parameter 330, API operating system message 360, and API resources 390 context information. Each of these portions of the input API call records is discussed herein below.

Please amend the paragraph beginning at page 19, line 28, to read as follows:

The API call records include API parameter information 330 for each parameter associated with the API call (both input and output). A size field specifies the length of the parameter (e.g., the number of bytes). The parameter type and base type are specified by both name and ID. In the case where the parameter is an array, an array size (e.g., char[260]) field stores the dimensions of the array. The API replay tool supports mapping pointer to actual values. The level of indirection is specified by an indirection field (e.g., DWORD* pdw has an indirection value of 1 while DWORD** pdw has an indirection value of 2). A modifier field specifies whether the parameter is an input parameter, output parameter or input/output parameter. The parameter information 330 also supports structure parameters (in the context of C/C++ programming a structure is a set of combined parameters). In the case of structures, the number of members and some form of directions for reading/enumerating them from the structure is designated. A GUID field (or fields) stores any unique identifiers assigned to the parameter. Yet another field or combination of fields specify parameters to be retrieved by GetNumParams, GetParam(ParamNum) calls associated with an API call parameter. Another set of fields specifies verification values from an extended manifest (e.g., GetValidRange, GetInvalidRange, GetAppValidParamRange, GetAppInvalidParamRange). Another parameter information field handles values attached to the parameters (e.g., enum var(EMPTY= 0, API, MSG) GetNumValues = 3, GetValueString(1) = API). Finally, an embodiment of the invention supports designating flags associated with each parameter through calls to the system (e.g., GetNumFlags, GetFlag).

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

Please amend the paragraph beginning at page 20, line 18, to read as follows:

The [[API]] operating system message ~~information~~ 360 includes a set of fields defining any operating system messages that arose from the logged API call. Each message is identified by a message ID and message type (i.e., user or system message). In addition to passing an LParam and WParam message parameter (information holders), a time stamp specifies when the message was issued by the operating system. A handle field specifies a unique handle assigned to the message for purposes of identifying the message within the context of the application executing on the system.

Please amend the paragraph beginning at page 22, line 19, to read as follows:

A process block interface 510 includes specialized methods for managing thread memory resources within a process block. Examples of such methods ~~included~~ include, for example: GetFirstThreadBlock, GetNextThreadBlock, GetPrevThreadBlock and GetThread which returns a thread block address corresponding to a specified thread ID.

Please amend the paragraph beginning at page 22, line 27, to read as follows:

A memory block interface 530 is a template for any of the extensible set of memory block types managed by the thread block interface 520. A code data block interface 540 managed memory is allocated for the output parameters of an API call of a particular thread. The code data block interface 540 is used by the symbol tables 214 to keep track of output ~~parameters~~ parameter values.

Please amend the paragraph beginning at page 23, line 1, to read as follows:

Having described the general organization of the API replay tool 200 and the functions performed by its primary components, attention is directed to **FIG. 6** that depicts [[and]] an exemplary sequence of steps performed by a system including the API replay tool to process a set of logged API calls. Initially, during step 600 the API replay tool 200 initializes, if it has previously not been active, in response to a call to [[the]] initialize method 400 on the replay engine 212. In addition to performing its own initialization, in an embodiment of the invention,

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206.682.8100

the replay engine 212 initializes other components of the API replay tool 200 (e.g., the log reader 210, thread handler 216, etc.).

Please amend the paragraph beginning at page 23, line 24, to read as follows:

At step 608 the API call builder 222 initially processes the memory references within the API call. In cases where the references require translation to the replay context, the API call builder 222 determines new addresses for referenced data structures and variables within a memory block allocated to a thread identified within the input API call record. The API call builder 222 adds mapping entries in the symbol tables 214 as needed to map addresses of parameters (e.g., pointers, variables, etc.) specified in the input API call record to corresponding memory locations assigned to the parameter values within the execution environment of the API replay tool 200. The API call builder thereafter generates an assembly code instruction sequence corresponding to the API call. The resulting API call includes the mapped memory addresses for the passed parameters. The API call builder passes a reference to the assembly code instruction sequence to the API call executer 224.

Please amend the paragraph beginning at page 24, line 4, to read as follows:

At step 610 the API call executer 224 executes assembly code instructions corresponding to the API call by invoking the appropriate system module (e.g., DLL) to carry out the function specified in the API call structure based upon the passed parameter data. Upon completion, the API call executer 224, at step 612, notifies the replay engine 212 (and passes any relevant completion data/error messages).

LAW OFFICES OF
CHRISTENSEN O'CONNOR JOHNSON KINDNESS PLLC
1420 Fifth Avenue, Suite 2800
Seattle, Washington 98101
206 682 8100